Gnutella Style Peer-to-Peer (P2P) File Sharing

A JAVA Application (Design Documentation)

by

Abhishek Kumar

Table of Contents

1. Project Statement 3

2. Design 3

3. Config file 5

4. Implementation 7

5. Trade-offs and Improvements 8

1. Project Statement

In the Gnutella-style peer-to-peer (P2P) file sharing system, each peer should act as both a server and a client. As a client, it provides interfaces through which users can issue queries and view search results. As a server, it accepts queries from other peers, checks for matches against its local data set, and responds with corresponding results. In addition, since there's no central indexing server (like Napster style peer-to-peer file sharing), search is done in a distributed manner. Each peer maintains a list of peers as its neighbor. Whenever a query request comes in, the peer will broadcast the query to all its neighbors in addition to searching its local storage (and responds if necessary).

2. Design

In the first design, peers are connected together as a star topology. In this system, peer one placed in the middle while the others connected statically to peer one.

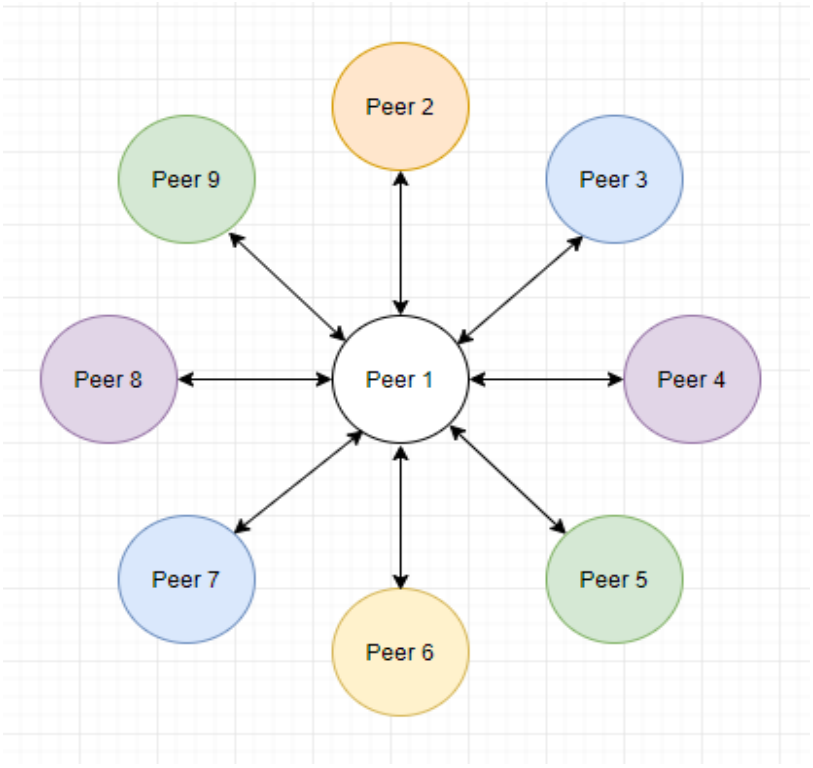

Fig 1: Star Topology

In the second design, peers are connected together as a mesh topology. In this system, peers are placed in a 2D-mesh one placed in the middle while the others connected statically to peer one.

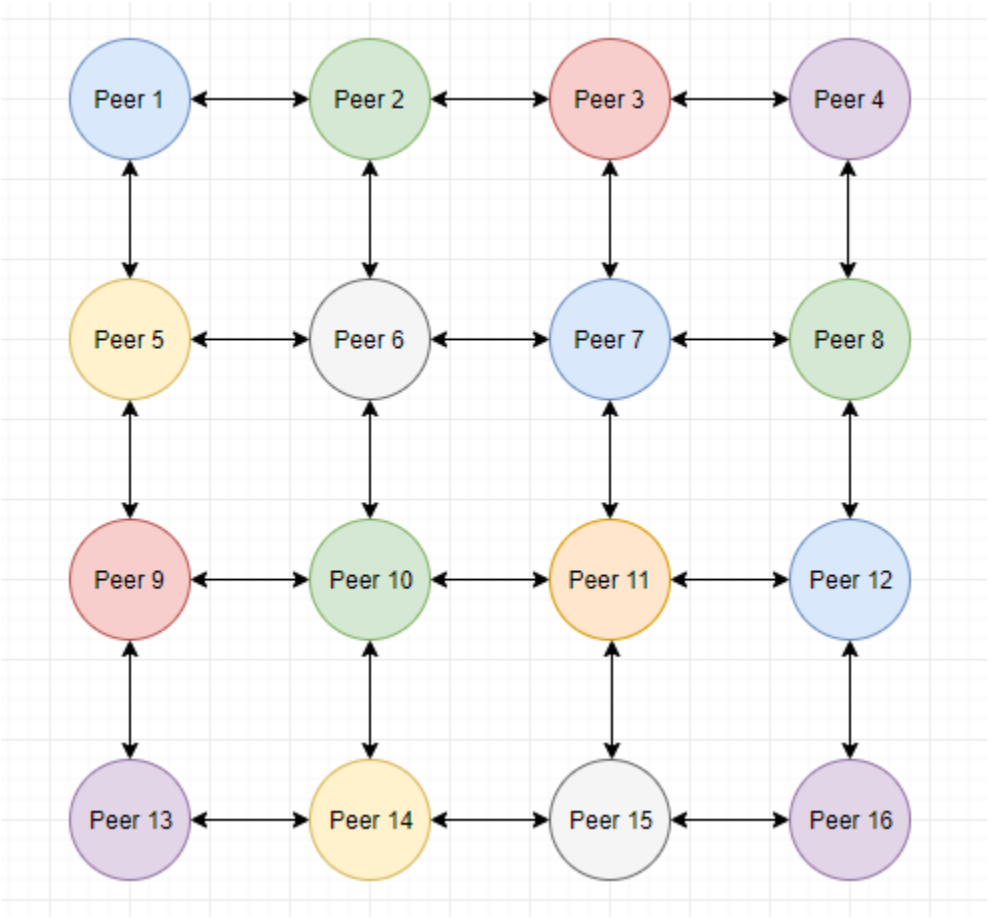

Fig 2: Mesh Topology

3. Config file

A full-featured Gnutella system is dynamically initialized. For our simplicity, we keep the structure of the P2P network as static. The neighbor peers are declared in the config file as shown below.

The star topology structure is predefined in a config.txt file in the system as following. These config.txt contains the peer IP and ports number. They can be changed if needed.

```
peer1.port=4000
peer2.port=4001
peer3.port=4002
peer4.port=4003
peer5.port=4004
peer6.port=4005
peer7.port=4006
peer8.port=4007
peer9.port=4008
port1.next=2,3,4,5,6,7,8
port2.next=1
port3.next=1
port4.next=1
port5.next=1
port6.next=1
port7.next=1
port8.next=1
port9.next=1

peer1.serverport=50000
peer2.serverport=50001
peer3.serverport=50002
peer4.serverport=50004
peer5.serverport=50005
peer6.serverport=50006
peer7.serverport=50007
peer8.serverport=50008
peer9.serverport=50009


```

The mesh topology structure is predefined in a config.txt file in the system as following. These config.txt contains the peer IP and ports number. They can be changed if needed.

```
peer1.port=4000
peer2.port=4001
peer3.port=4002
peer4.port=4003
peer5.port=4004
peer6.port=4005
peer7.port=4006
peer8.port=4007
peer9.port=4008
peer10.port=4009
peer11.port=4010
peer12.port=4011
peer13.port=4012
peer14.port=4013
peer15.port=4014
peer16.port=4015
```

```
port1.next=2,5
port2.next=1,3,6
port3.next=2,4,7
port4.next=3,8
port5.next=1,6,9
port6.next=2,5,7,10
port7.next=3,6,8,11
port8.next=4,12
port9.next=5,10,13
port10.next=6,9,11,14
port11.next=7,10,12,15
port12.next=8,16
port13.next=9,14
port14.next=10,13,15
port15.next=11,14,16
port16.next=12,15
```

```
peer1.serverport=50000
peer2.serverport=50001
peer3.serverport=50002
peer4.serverport=50004
peer5.serverport=50005
peer6.serverport=50006
peer7.serverport=50007
peer8.serverport=50008
peer9.serverport=50009
peer10.serverport=50009
peer11.serverport=50010
peer12.serverport=50011
peer13.serverport=50012
peer14.serverport=50013
peer15.serverport=50014
peer16.serverport=50015
```

4. Implementation

In this Project, I have extensively used the concept of multithreading. For every peer that goes online, a server thread is started which runs in the background. Then the *config.properties* file is read to find the neighbors of current peer and starts a client thread for each neighbor. When the neighboring peer gets online a connection is established between client thread of previous peer and server thread of neighboring peer. Using this connection file search among the peers is performed.

ServerThread.java is used to search for the file within its local directory and store the result in an array. Also once the local search is performed it reads the config.properties file to start a client thread for all its neighbors.

ClientThread.java is used to send the file to be searched to its connected server. It also provides a method to forward to all its upstream neighbors an array of peers containing file.

ServerDownload.java thread performs the downloading of file from the peer chosen by the user once all peers with files are listed to the user by establishing a server & client connection.

Working of the file ***Main.java***

With the execution of the Main.java class user enters the peer-id of the currently executing peer and its local directory. Then a thread instance of Server thread is created which runs in the background waiting for a Client thread with same port no read from the configuration file.
Next, peer reads peer-ids of all its neighbors and starts a Client thread for all its neighbors which wait for Server thread with the same respective port numbers. The further execution of current peer waits till the completion of all the client thread is done. Once all client threads completed, for each thread we retrieve the array containing the peers with file. All peers with files are then listed to the user. The user selects the peer from which the file is to download. Once the peer-id is retrieved the ServerDownload.java thread is invoked which acts as Server port to which the connection is established as a Client. Then the requested file is downloaded as stream of bytes of data is sent from Server to Client. Once the downloading is done the searched file is downloaded in local directory of the peer that had requested for the file.

5. Trade-offs and Improvements

Implementation of sockets makes this Client Server model platform independent, therefore a Client and Server would form a connection regardless of the platform in which they are implemented. But in usage, when a client connects to the server till that time server continuously polls for request from a client leading to low performance. Also, since the threads are extensively used hence it induces a delay when there are thousands of peer connected together in a topology.
Also the InputStream of client continuously polls until it receives a data from client.